Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation

Ian Foster[1,2] Jens Vöckler[2] Michael Wilde[1] Yong Zhao[2]
[1] Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA
[2] Department of Computer Science, University of Chicago, Chicago, IL 60637, USA
{foster,wilde}@mcs.anl.gov, {voeckler,yongzh}@cs.uchicago.edu

Abstract

Much scientific data is not obtained from measurements but rather derived *from other data by the application of computational procedures. We hypothesize that explicit representation of these procedures can enable documentation of data provenance, discovery of available methods, and on-demand data generation (so-called "virtual data"). To explore this idea, we have developed the* Chimera virtual data system, *which combines a virtual data catalog, for representing data derivation procedures and derived data, with a virtual data language interpreter that translates user requests into data definition and query operations on the database. We couple the Chimera system with distributed "Data Grid" services to enable on-demand execution of computation schedules constructed from database queries. We have applied this system to two challenge problems, the reconstruction of simulated collision event data from a high-energy physics experiment, and the search of digital sky survey data for galactic clusters, with promising results.*

1 Introduction

In many scientific disciplines, the analysis of "data" (whether obtained from scientific instruments, such as telescopes, colliders, or climate sensors, or from numerical simulations) is a significant community activity. As a result of this activity, communities construct, in a collaborative fashion, collections of derived data (e.g., flat files, relational tables, persistent object structures) with relationships between data objects corresponding to the computational procedures used to derive one from another (Figure 1). Recording and discovering these relationships can be important for many reasons, as illustrated by the following vignettes.

"I've come across some interesting data, but I need to understand the nature of the corrections applied when it was constructed before I can trust it for my purposes."

"I want to search an astronomical database for galaxies with certain characteristics. If a program that performs this analysis exists, I won't have to write one from scratch."

"I want to apply an astronomical analysis program to millions of objects. If the program has already been run and the results stored, I'll save weeks of computation."

"I've detected a calibration error in an instrument and want to know which derived data to recompute."

"I want to find those results that I computed last month, and the details of how I generated them."

More generally, we want to be able to track how data products are derived—with sufficient precision that one can create and/or re-create data products from this knowledge. One can then explain definitively how data products are created, something that is often not feasible even in carefully curated databases. One can also implement a new class of "virtual data management" operations that, for example, "re-materialize" data products that were deleted, generate data products that were defined but never created, regenerate data when data dependencies or transformation programs change, and/or create replicas of data products at remote locations when re-creation is more efficient than data transfer.

In order to explore the benefits of data derivation tracking and virtual data management, we have designed, prototyped, and experimented with a virtual data system called *Chimera*. A *virtual data catalog* (based on a relational *virtual data schema*) provides a compact and expressive representation of the computational procedures used to derive data, as well as invocations of those procedures and the datasets produced by those invocations. A *virtual data language interpreter* executes requests for constructing and querying database entries.

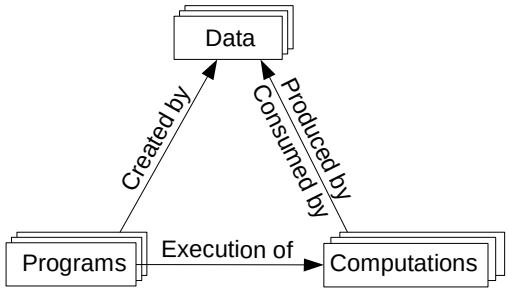

Figure 1: Relationships among programs, computations, and data

In: *14th Intl. Conf. on Scientific and Statistical Database Management, Edinburgh, July, 2002.*

We couple Chimera with other Data Grid services [1, 11, 18, 28] to enable the creation of new data by executing computation schedules obtained from database queries, and the distributed management of resulting data.

We have applied the Chimera system successfully to two challenging physics data analysis computations, one involving the generation and reconstruction of simulated high-energy physics collision event data from the Compact Muon Solenoid (CMS) experiment at CERN [20, 25], and the other the detection of galactic clusters in Sloan Digital Sky Survey (SDSS) data [4, 29]. Our results demonstrate our ability to track data derivations and to schedule large distributed computations in response to user virtual data queries. Others have successfully used some of these techniques in the analysis of data from the LIGO gravitational wave observatory [14].

The importance of being able to document provenance is well known [30]. Our work builds on preliminary explorations within the GriPhyN project [5, 15, 16]. There are also relationships to work in database systems [9, 10, 31] and versioning [8, 26]. Cui and Widom [12, 13] record the relational queries used to construct materialized views in a data warehouse, and then exploit this information to explain lineage. Our work can leverage these techniques, but differs in two respects: first, data is not necessarily stored in databases and the operations used to derive data items may be arbitrary computations; second, we address issues relating to the automated generation and scheduling of the computations required to instantiate data products.

Early work on *conceptual schemas* [21] introduced virtual attributes and classes, with a simple constrained model for the re-calculation of attributes in a relational context. Subsequent work produced an integrated system for scientific data management called ZOO [22], based on a special-purpose ODBMS that allowed for the definition of "derived" relationships between classes of objects. In ZOO, derivations can be generated automatically based on these relationships, using either ODBMS queries or external transformation programs. Chimera is more specifically oriented to capturing the transformations performed by external programs, and does not depend on a structured data storage paradigm or on fine-grained knowledge of individual objects that could be obtained only from an integrated ODBMS.

We can also draw parallels drawn between Chimera and workflow [23, 27] and knowledge management systems that allow for the definition, discovery, and execution of (computational) procedures.

The rest of this article is as follows. We first introduce the Chimera virtual data system and describe its virtual data schema and language (Sections 2-4). Then, we discuss the integration of Chimera with Data Grids (Section 5), our experiences applying the system to challenge problems (Section 6), and future directions.

2 Chimera Architecture

The architecture of the Chimera virtual data system is depicted in Figure 2. In brief, it comprises two principal components: a *virtual data catalog* (VDC; this implements the Chimera *virtual data schema*) and the *virtual data language interpreter*, which implements a variety of tasks in terms of calls to virtual data catalog operations.

Applications access Chimera functions via a standard *virtual data language* (VDL), which supports both *data definition* statements, used for populating a Chimera database (and for deleting and updating virtual data definitions), and *query* statements, used to retrieve information from the database. One important form of query returns (as a directed acyclic graph, or DAG) a representation of the tasks that, when executed on a Data Grid, create a specified data product. Thus, VDL serves as a lingua franca for the Chimera virtual data grid, allowing components to determine virtual data relationships, to pass this knowledge to other components, and to populate and query the virtual data catalog without having to depend on the (potentially evolving) catalog schema.

Chimera functions can be used to implement a variety of applications. For example, a *virtual data browser* might support interactive exploration of VDC contents, while a *virtual data planner* might combine VDC and other information to develop plans for computations required to materialize missing data (Section 5).

The Chimera virtual data schema defines a set of relations used to capture and formalize descriptions of how a program can be invoked, and to record its potential and/or actual invocations. The entities of interest—transformations, derivations, and data objects—are as follows; we describe the schema in more detail below.

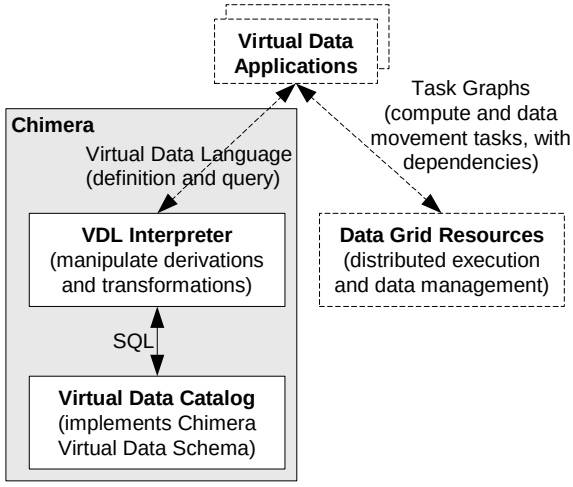

Figure 2: Schematic of the Chimera architecture

- A *transformation* is an executable program. Associated with a transformation is information that might be used to characterize and locate it (e.g., author, version, cost) and information needed to invoke it (e.g., executable name, location, arguments, environment).
- A *derivation* represents an execution of a transformation. Associated with a derivation is the name of the associated transformation, the names of data objects to which the transformation is applied, and other derivation-specific information (e.g., values for parameters, time executed, execution time). While transformation arguments are *formal* parameters, the arguments to a derivation are *actual* parameters.
- A *data object* is a named entity that may be consumed or produced by a derivation. In the applications considered to date, a data object is always a *logical file*, named by a *logical file name* (LFN); a separate *replica catalog* or *replica location service* is used to map from logical file names to physical location(s) for replicas [2, 11]. However, data objects could also be relations or objects. Associated with a data object is information about that object: what is typically referred to as metadata.

We do not address here the question of how the data dependency information maintained within the Chimera system is produced. Information about transformations and derivations can potentially be declared explicitly by the user, extracted automatically from a job control language, produced by higher-level job creation interfaces such as portals, and/or created by monitoring job execution facilities and file accesses.

Information can be recorded in the virtual data system at various times and for various purposes. Transformation entries generated before invocation can be used to locate transformations and guide execution. Derivation entries generated before jobs are executed can provide information needed to generate a file. Entries generated *after* a job is executed record how to *re*generate a file. Consider, for example, transformation entries that define the four stages of a simulation pipeline. An initial query for the output of this pipeline returns a DAG that, when executed, generates files called `f1 - f4`. Subsequent deletion of file `f3` followed by a retrieval request for that file results only in the re-execution of stage 3 of the pipeline.

3 Chimera Virtual Data Schema

We describe here the Chimera virtual data schema, shown in Figure 3.

A logical transformation is characterized by its identifying name, the namespace within which the name is unique, and a version number. The signature of the transformation includes input and output parameters, which need not be files. A transformation may have an arbitrary number of formal arguments. Thus the relationship between `TRANSFORMATION` and `FORMALARG` is 1:N.

A transformation may have more than one derivation, each supplying different values for the parameters. A derivation may be applicable to more than one transformation. Thus, versioning allows for a range of valid transformations to apply, increasing the degrees of freedom for schedulers to choose the most applicable one.

An `ACTUALARG` relates to a derivation. Its value captures either the LFN or the value of a non-file parameter. A `FORMALARG` may contain an optional default value, captured in a similar fashion by the same `VALUE` class. The `VALUE` class is an abstract base class for either a single value (`SCALAR`) or a list of similar values (`LIST`), which are collapsed union-fashion into a single table.

The relationships between a transformation and its formal parameters, on the one hand, and a dependent derivation and its actual parameters, on the other, are not independent of each other. Each instantiation of an actual parameter maps to exactly one formal parameter describing the entry. The binding is created using the argument name, not its position in the argument list.

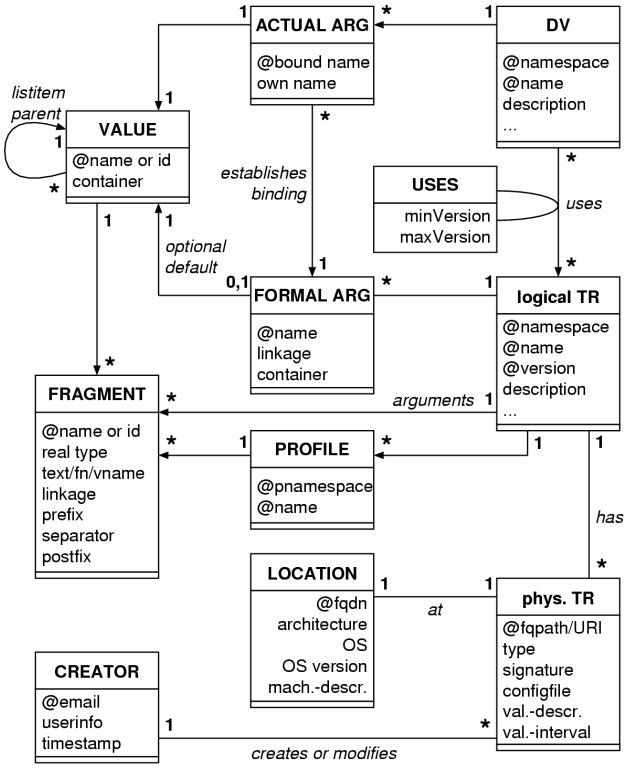

Figure 3: UML description of the Chimera schema

The arguments on the command line of a transformation are captured in multiple fragments, each either a reference to a formal argument, or a textual string.

Scheduler- and runtime environment-specific data is abstracted in the PROFILE table. For example, in the case of a Unix environment variable, the namespace is "env", the key within this namespace is the environment variable name, and the value is a list of fragments, either references to bound variables or textual strings.

The FRAGMENT table captures three child classes, either a textual string, a LFN or a reference to a bound variable. The three child classes are collapsed into a single table.

The lower portion of the diagram deals with the physical location of any given transformation [16].

Finally, we note that Chimera applications will typically also require a METADATA table, which maps from (key, value) attribute pairs to LFNs, and a REPLICA table, which maps from LFNs to physical file locations. However, these relations are frequently implemented via a separate metadata catalog [7] and/or replica catalog [11] and so they are not considered here.

4 Chimera Virtual Data Language

As noted above, the Chimera *virtual data language* (VDL) comprises both data definition and query statements. We first introduce two data derivation statements, TR and DV, and then discuss queries. While our implementation uses XML internally, we use a more readable syntax here.

4.1 The TR Data Definition Statement

A TR statement defines a transformation. When the VDL interpreter processes such a statement, it creates a transformation object within the virtual data catalog. For example, the following definition provides the information required to execute a program app3.

```
TR t1( output a2, input a1,
       none env="100000",
       none pa="500" ) {
  app vanilla = "/usr/bin/app3";
  arg parg = "-p "${none:pa};
  arg farg = "-f "${input:a1};
  arg xarg = "-x -y ";
  arg stdout = ${output:a2};
  profile env.MAXMEM = ${none:env};
}
```

This definition reads as follows. The first line assigns the transformation a name (t1) for use by derivation definitions, and declares that t1 reads one input file (formal parameter name a1) and produces one output file (formal parameter name a2). The parameters declared in the TR header line are *transformation arguments* and can only be file names or textual arguments.

The APP statement specifies (potentially as an LFN) the executable that implements the execution.

The first three ARG statements describe how the *command line arguments* to app3 (as opposed to the *transformation arguments* to t1) are constructed. Each ARG statement comprises a name (here, parg, farg, and xarg) followed by a default value, which may refer to transformation arguments (e.g., a1) to be replaced at invocation time by their value. The special argument stdout (the fourth ARG statement in the example) is used to specify a filename into which the standard output of an application would be redirected.

Argument strings are concatenated in the order in which they appear in the TR statement to form the command line. The reason for introducing argument names is that these names can be used within DV statements to override the default argument values specified by the TR statement.

Finally, the PROFILE statement specifies a default value for a Unix environment variable (MAXMEM) to be added to the environment for the execution of app3.

4.2 The DV Data Definition Statement

A DV statement defines a derivation. When the VDL interpreter processes such a statement, it records a transformation invocation within the virtual data catalog. A DV statement supplies LFNs for the formal filename parameters declared in the transformation and thus specifies the *actual* logical files read and produced by that invocation. For example, the following statement records an invocation of transformation t1 defined above.

```
DV t1(
  a2=@{output:run1.exp15.T1932.summary},
  a1=@{input:run1.exp15.T1932.raw},
  env="20000", pa="600" );
```

The string immediately after the DV keyword names the transformation invoked by the derivation.

In contrast to transformations, derivations need not be named explicitly via VDL statements. They can be located in the catalog by searching for them via the logical filenames named in their IN and OUT declarations as well as by other attributes, as discussed below.

Actual parameters in a derivation and formal parameters in a transformation are associated by name. For example, the statements above result in parameter a1 of t1 receiving the value run1.exp15.T1932.raw and a2 the value run1.exp15.T1932.summary.

The example DV definition corresponds to the following invocation:

```
export MAXMEM=20000
/usr/bin/app3 -p 600 \
    -f run1.exp15.T1932.raw -x -y \
    > run1.exp15.T1932.summary
```

Filenames listed as `IN` and `OUT` in a transformation need not necessarily appear as command line arguments in a corresponding derivation. For example, if a filename was determined directly by the executable through an internal definition or is determined dynamically, it might not appear on the command line even though the file is read or written by the application. In such cases, applications that know or can detect what filenames were read and written by an application could, after the fact, create a derivation record to describe these dynamic data dependencies.

The filename insertion semantics described here support a wide variety of argument passing conventions. Executables with argument passing conventions that cannot be expressed in these terms must be executed by creating "wrapper" scripts or executables that adapt the VDL conventions to those expected by the executable. For example, applications that read the names of further input files from a "control" file can often be handled with a wrapper that accepts the filenames as command line arguments and places them in the control file before calling the actual application. In some cases, the creation of the control file itself could be described as a transformation that reads several files and produces the control file as its output. Then the real transformation can be described as having the control file as an input.

4.3 Tracking Derivation Dependencies

Chimera's VDL supports the tracking of data dependency chains among derivations. For example, the following statements define two derivations (as well as two transformations), such that the output of the first is the input to the second. Thus, we can conclude that (unless `file2` or `file3` exist) to generate `file3` we must run `trans1` before `trans2`, and use its output file (`file2`) as the input file for `trans2`.

```
TR trans1( output a2, input a1 ) {
  app vanilla = "/usr/bin/app1";
  arg stdin = ${input:a1};
  arg stdout = ${output:a2};
}
TR trans2( output a2, input a1 ) {
  arg vanilla = "/usr/bin/app2";
  arg stdin = ${input:a1};
  arg stdout = ${output:a2};
}
DV trans1( a2=@{output:file2},
  a1=@{input:file1} );
DV trans2( a2=@{output:file3},
a1=@{input:file2} );
```

We can thus construct arbitrarily complex directed acyclic execution graphs (DAGs) automatically. For example, consider the transformations illustrated in Figure 4. Four logical files, named `f.a`, `f.b`, `f.c`, and `f.d` in the figure, are produced as a result of this computation. The output from a first node `generate` is stored into file `f.a`. Two processes `findrange` each operate on disjoint subsets of the input `f.a`, publishing their results in `f.b` and `f.c`, respectively. A final node, `analyze`, combines the two halves.

The following statements define the transformations `generate`, `findrange`, and `analyze`, and the derivations that produce files `f.a`, `f.b`, `f.c`, and `f.d`.

```
TR generate( output a ) {
  app vanilla = "generator.exe";
  arg stdout = ${output:a2};
}
TR findrange( output b, input a,
   none p="0.0" ) {
  app vanilla = "ranger.exe";
  arg arg = "-i "${:p};
  arg stdin = ${output:a};
  arg stdout = ${output:b};
}
TR analyze( input a[], output c ) {
  app vanilla = "analyze.exe";
  arg files = ${:a};
  arg stdout = ${output:a2};
}
DV generate( a=@{output:f.a} );
DV findrange( b=@{output:f.b},
  a=@{input:f.a}, p="0.5" );
DV findrange( b=@{output:f.c},
  a=@{input:f.a}, p="1.0" );
DV analyze( a=[ @{input:f.b},
  @{input:f.c} ], c=@{output:f.d} );
```

Notice that the transformation `findrange` is invoked twice, with different values for the command line argument `-i` specifying different search ranges.

4.4 Compound Transformations

A *compound transformation* describes the coordinated (perhaps concurrent) execution of multiple programs and the passing of files among them. It is described in the same manner as a simple transformation, with a single derivation statement. All internal transformation invocations within a compound transformation are tracked in the catalog, along with all files read and produced by the internal transformation steps. The system can thus remain fully cognizant of all data dependencies, and arbitrary files within those dependency chains can be deleted and later re-derived, based on stored knowledge.

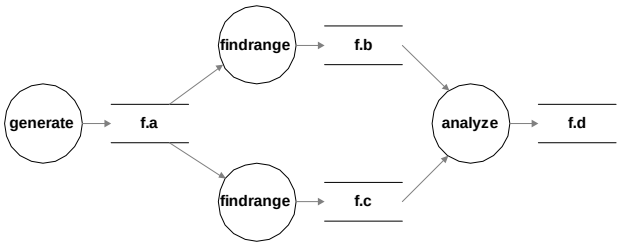

Figure 4: Example directed acyclic graph

A transformation is either a simple transformation or a compound transformation. A compound transformation is itself composed of references to one or more transformations, each of which in turn are either simple or compound. In all other respects, and in particular, from the point of view of its external interface and semantics, compound and simple transformations are indistinguishable. Thus, compound transformations can themselves contain compound transformations.

4.5 Queries

VDL provides various commands for extracting derivation and transformation definitions. Since VDL is implemented in SQL, this query set is readily extensible. Query output can (optionally) be returned in the same format as the commands that could be used to re-create the matching entries. Query commands can be used both by end-user query systems (e.g., a virtual data browser) and by automated Grid components such as a data analysis system.

In brief, VDL query commands allow one to search for *transformations* by specifying a transformation name, application name, input LFN(s), output LFN(s), argument matches, and/or other transformation metadata. One can search for *derivations* by specifying the associated transformation name, application name, input LFN(s), and/or output LFN(s). An important search criterion is whether derivation definitions exist that invoke a given transformation with specific arguments. From the results of such a query, a user can determine if desired data products already exist in the data grid, and can retrieve them if they do and create them if they do not.

Query output options specify, first of all, whether output should be recursive or non-recursive (recursive shows all the dependent derivations necessary to provide input files assuming no files exist) and second whether to present output in columnar summary, VDL format (for re-execution), or XML.

5 Chimera as a Data Grid Component

We discuss some of the issues that arise when the Chimera system is incorporated as a component within a larger Data Grid system. As illustrated in Figure 2, a virtual data "application" can combine information from both Chimera and other Data Grid components as it processes user requests for virtual data. For example, an application might combine information about the materialization status of a requested derivation with information about the physical location of replicas and the availability of computing resources to determine whether to access a remote copy or (re-)generate a data value.

One proposed Data Grid architecture [17] interposes a planner between an application and other components illustrated in Figure 2. The planner accepts *abstract DAGs* from the application, that is, DAGs that refer only to LFNs and not to specific physical instances of files, and that are thus not yet bound to specific Grid locations. For LFNs needed as input, an abstract DAG does not specify if these files already exist at a computation site, need to be copied there, or should be re-derived; for LFNs produced as output, the planner must determine where to place the newly created file. Location decisions must also be made recursively for any additional derivations needed.

The planner examines the abstract DAG, selects an execution site for each node, and then determines how to obtain and transport the data needed by each computation. The planner must also determine how to deal with the relocation of physical files *produced* by a job, if policies require that these files be relocated to specific physical file storage servers. The planner may evaluate several different execution plans, based for example on cost estimates for data movement vs. re-creation.

The output of the request planner is a *concrete* DAG that refers only to real physical file names and specifies the steps that must be followed to compute or transport any input data that does not yet exist at its execution site.

6 Experiences with the Chimera System

We describe application experiments with our Chimera prototype, conducted on the small-scale Data Grid shown in Figure 5. (Subsequent experiments will use the larger International Virtual Data Grid Laboratory [6].) This Grid used Globus Toolkit resource management and data transfer components [17], Condor schedulers and agents [19, 24], and the DAGman job submission agent to coordinate resources at four sites.

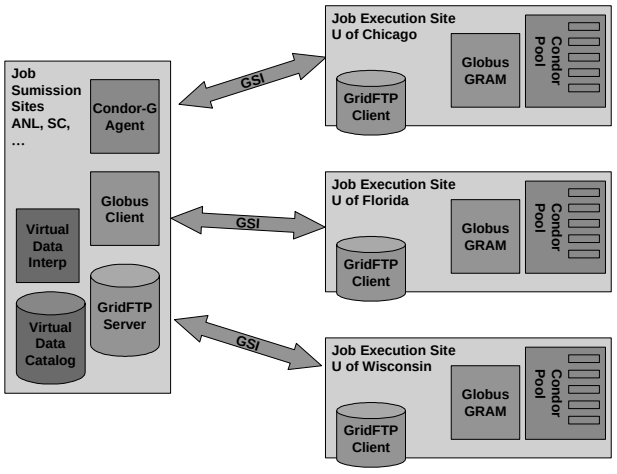

Figure 5: The Data Grid used in our experiments

In our experiments, we did not consider data replication. The persistent location of all data products was the site at which the VDL interpreter resided. All data files needed by executables were pulled to the executing sites, and all data files from successful executions were returned to the submission site upon completion of data derivations. All data transfers used the Grid-enabled data transfer tool GridFTP [2].

As we explain, the results demonstrate that Chimera can manage complex interdependencies among application invocations that occur in practice. We have also validated the capability of the Chimera system on a variety of larger and more complex artificial DAGs.

6.1 CMS Data Reconstruction

We used the Chimera prototype to assess the feasibility of using virtual data descriptions for data products involved in the production of Monte Carlo-based simulations of high-energy physics collision events in the CMS experiment [20].

Event simulation is critical to the design and operation of the complex detector that is at the heart of CMS, and is also used to test the scientific and data management software systems on which CMS will depend.

This complex physics data derivation process comprises the following four-stage pipeline of transformations (i.e., executables).

1. **pythia** using Monte Carlo techniques to determine randomly the physics attributes of a specific collision event.
2. **cmsim** determines how that event would affect the CMS detector.
3. **writehits** converts the information into a persistent object data structure in an object-oriented database system.
4. **writedigis** determines the digital signals that the detector would produce from the event.

The first two stages produce files as output, while the second two stages produce object-oriented databases that are contained in the files of an Objectivity federation.

In these experiments, we sidestepped many of the complications of managing data stored in object form by configuring simulations to produce just one event per file, instead of the usual hundreds. However, we did use the Objectivity database used in normal CMS simulation.

6.2 SDSS Galactic Structure Detection

Our second Chimera application concerns the analysis of data from the Sloan Digital Sky Survey (SDSS) [29]. As described at www.sdss.org, "…the Sloan Digital Sky Survey is the most ambitious astronomical survey project ever undertaken. The survey will map in detail one-quarter of the entire sky, determining the positions and absolute brightness of more than 100 million celestial objects. It will also measure the distances to more than a million galaxies and quasars." The project, which will survey the night sky at an unprecedented resolution, is currently in its third year of data taking. When complete in 2004 it will have collected around 40 TB of image and spectroscopic data, and 3TB of catalog metadata.

Working with collaborators at Fermilab, we applied the concept of virtual data to one scientific challenge problem on the SDSS project—that of locating galactic clusters in the image collection [3]. The goal of this application is to create a database of galaxy clusters for the entire survey. A highly simplified view of this problem is as follows.

The sky is tiled into a set of regular "fields." For each field, clusters are searched for in that field and in some set of neighboring fields, using the concepts of "brightest cluster galaxy" (BCG) and "brightest red galaxy" (BRG) to determine cluster candidates [3]. The resulting algorithm is a tree-structured pipeline (Figure 6) comprising the following five transformations.

1. **fieldPrep** extracts from the full data set required measurements on the galaxies of interest and produces new files containing this data. The new files are about 40 times smaller than the full data sets.
2. **brgSearch** calculates the unweighted BCG likelihood for each galaxy. The unweighted likelihood may be used to filter out unlikely candidates for the next stage.
3. **bcgSearch** calculates the weighted BCG likelihood for each galaxy. This is the heart of the algorithm, and the most expensive step.
4. **bcgCoalesce** determines whether a galaxy is the most likely galaxy in the neighborhood.
5. **getCatalog** removes extraneous data and stores the result in a compact format.

Further details of the algorithms and astrophysics mentioned here are provided elsewhere [3, 4, 32].

Figure 7 shows the actual dataflow found in the last three stages of a small computation in which 24 *brgSearch* transformations (the leaves) reduce 156 files down to the root, where the *getCatalog* transformation produces the cluster catalog for a single field of the sky.

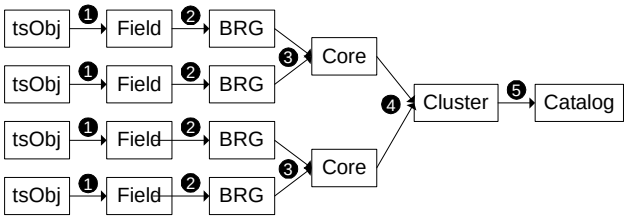

Figure 6: SDSS cluster identification workflow.

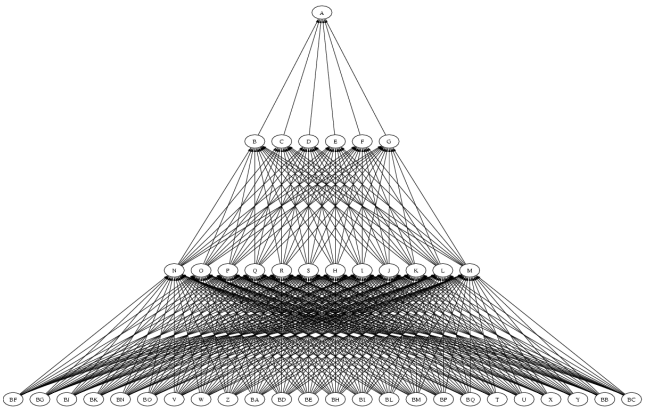

Figure 7: DAG for cluster identification workflow.

The derivation *getCatalog* now becomes a function that can invoke all the prior four dependent steps. To generate "virtual" results for the entire sky, we define one derivation of the *getCatalog* transformation for each field of the current survey.

This application motivated an important extension to the virtual data paradigm to permit the specification of transformations where the input and output file names are a function of one or more transformation arguments and thus cannot be known at the time when the transformation or even the derivation are defined. In the cluster finding mechanism, the celestial coordinates of the neighboring fields that will be required to locate the clusters within a given field of the survey are a function of that central field's coordinates—i.e., they are the neighbors of that central field, and their coordinates and hence their file names must be computed dynamically once the coordinates of the central field are known. Table 1 characterizes the five transformations used in the pipeline in terms of the number of static and dynamic file names.

Table 1: Input and output files for galactic cluster identification computation

Transform	Fixed files in	No. of varying input lists & files/list	Fixed files out	Varying output lists
fieldPrep	1	1 (13*12)		1 (13*12)
brgSearch	1	1 (13*12)		1 (13*12)
bcgSearch	1	2 (13*12,13*12)		1 (7*12)
bcgCoalesce	1	1 (7*12)		1 (12)
getCatalog	1	1 (12)	1	

Table 2: Numbers of files produced and consumed for galactic cluster finding on entire sky survey

Transform	# derivations	#files in	#files out
fieldPrep	(45x600x12)/10	45x600x12	45x600x12
brgSearch	(45x600x12)/10	45x600x12	45x600x12
bcgSearch	(45x600x12)/10	**2x**45x600x12	45x600x12
bcgCoalesce	(45x600x12)/10	45x600x12	45x600x12
getCatalog	(45x600)/10	45x600x12	45x600
Totals	**132,300**	**1,944,000**	**1,323,000**

The amount of computation and data access required to perform a complete analysis of all SDSS data is substantial. Table 2 summarizes the number of files that would be produced and consumed. At the time of writing, just over 2% of this data (one of the 45 eventual stripes of the sky) has been entered into the virtual data catalog as part of our experiments. The largest DAG executed to date for this application contained over 700 nodes. We have just started the larger computations required to complete the process, but it is already clear that the virtual data system simplifies the task tremendously. Without Chimera, the bookkeeping required to track a production effort of this magnitude would be considerable, and would involve a large amount of custom programming.

The virtual data mechanism can be thought of as a paradigm for the management of batch job production scripts. The Chimera system and its underlying Grid mechanisms automate all resource scheduling, synchronization, data movement, bookkeeping, and retry needed to manage the this large amount of work on a loosely coupled set of distributed resources.

The mechanism can also be thought of as a "makefile" for data production. For example, if the program *bcgCoalesce* is changed, a data administrator can request the re-creation of the final catalogs and Chimera will determine that 35,100 out of the total of 132,300 jobs need to be re-executed.

Similarly, if a few hundred of the 45x600x12 raw input files need to be revised (due to, say, errors discovered in the input capture system), then it is easy to determine what must be recomputed to update the entire output set.

It is also interesting to note that data production can be performed in parallel with interactive use by users who are requesting final output data products. In this mode, the virtual data grid acts much like a large-scale cache. If data products are produced through the batch process before they are needed interactively, then the system knows, at the time of the interactive request, that a data product is already available and no computations need be scheduled to produce it. If on the other hand a data product is requested before the batch process has produced it, the required derivations will be executed on demand to support the interactive need and will then be skipped when the batch process encounters a similar need at a later point in time.

7 Future Directions

We briefly discuss concepts that we plan to explore in future versions of the Chimera system.

Three data representation modes are dominant within GriPhyN experiments: files, relational tables (in RDBMSs) and persistent object structures (in ODBMSs). In addition, XML is proliferating as a universal and perhaps unifying underlying data representation. We believe that the mechanisms we have developed for file-based transformations and derivations extend naturally to encompass these other data modalities. If we can identify encapsulated data units, name these units, and specify a name and a pointer to transformations on the data, then our system can catalog both transformations and derivations in these data representation modes.

We define a unit of granularity that represents the entity that can be tracked: a file; an entire SQL table; or an entire object structure or "closure" that can be identified within the OODB and extracted, deleted, or regenerated as an atomic unit. Transformations described by Chimera can be executable programs, SQL command-level queries (including stored procedure invocations), ODBMS command-level queries or method invocations, or applications that access a SQL or OO database directly.

We believe that within this model it should be feasible to represent code transformations that freely exchange and transform data (within certain restrictive and well-defined limits) between these three modes of data storage. We plan to test this hypothesis on various challenge problems from the GriPhyN experiments.

We plan to augment the transformation description with information about the nature and state of the software and hardware environment in which a transformation executes [16]. This information can extend into the realm of configuration management systems.

We will also assess the utility of providing data type-based transformation templates—for example, to specify a transformation that translates "raw event data" files to "reconstructed event data" files, much as a makefile specifies a rule for translating a ".c" file into a ".o" file.

We will continue to explore the range of operations supported by the Chimera VDL, with the goal of validating our ability to realize the full spectrum of scenarios presented in the introduction.

Another significant research goal is to develop and test higher-level knowledge-based representations of domain-specific data, and to create databases and tools for representing and manipulating this knowledge. The question of how to support discovery of data, derivations, and transformations in a uniform fashion raises many challenging problems.

8 Conclusions

We have described Chimera, a virtual data tracking and generation system that can be used to audit and trace the lineage of derived data produced by computation and also to manage the automatic, on-demand (re)derivation of such data. This system comprises a relational database schema used to represent the various entities involved in data derivation, a virtual data language used to represent derivations and to manage the virtual data database, and a virtual data system used to manage the virtual data derivation process in large distributed data grid systems.

While the value of on-demand data derivation remains to be demonstrated in the general case, the value of auditing and tracing the lineage of scientific data in a large collaboration appears clear, as evidenced by the commitment of the four groundbreaking science experiments that comprise the Grid Physics Network. In general, we believe that virtual data techniques can significantly increase the usability of scientific data management systems by permitting science users to search for data based on application-level characteristics and automatically request the derivation of the data from pre-stored algorithm descriptions and derivation "recipes."

We have achieved positive results in our first tests of the Chimera system. These tests involved the automatic derivation of collider event simulation data in an application relating to the CMS high energy physics experiment, and automatic invocation of galactic cluster finding algorithms on SDSS data. We have also used various artificial problems to demonstrate our ability to manage more complex data derivation relationships.

These initial results encourage us that the Chimera design is viable and that it is feasible not only to represent complex data derivation relationships but also to integrate virtual data concepts into the operational procedures of large scientific collaborations. Further studies will provide additional insights into the utility of our techniques. We also plan to investigate the derivation of relational and object data, and the integration of higher-level techniques for representing ontologies.

Acknowledgements

We gratefully acknowledge helpful discussions with Rick Cavanaugh, Peter Couvares, Ewa Deelman, Greg Graham, Carl Kesselman, Miron Livny, and Alain Roy. This research was supported in part by the National Science Foundation under contract ITR-0086044 (GriPhyN), and by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38 (Data Grid Toolkit).

References

1. The DataGrid Architecture. EU DataGrid Project document DataGrid-12-D12.4-333671-3-0, 2001, www.eu-datagrid.org.
2. Allcock, W., Bester, J., Bresnahan, J., Chervenak, A.L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D. and Tuecke, S., Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. In *Mass Storage Conference*, (2001)
3. Annis, J., Kent, S., Castander, F., Eisenstein, D., Gunn, J., Kim, R., Lupton, R., Nichol, R., Postman, M. and Voges, W., The MaxBCG Technique for Finding Galaxy Clusters in SDSS Data. In *AAS 195th Meeting*, (2000)
4. Annis, J., Zhao, Y., Voeckler, J., Wilde, M., Kent, S. and Foster, I. Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey. Technical Report GriPhyN-2002-05, 2002, www.griphyn.org.
5. Avery, P. and Foster, I. The GriPhyN Project: Towards Petascale Virtual Data Grids. Technical Report GriPhyN-2001-15, 2001, www.griphyn.org.
6. Avery, P., Foster, I., Gardner, R., Newman, H. and Szalay, A. An International Virtual-Data Grid Laboratory for Data Intensive Science. Technical Report GriPhyN-2001-2, 2001, www.griphyn.org.
7. Baru, C., Moore, R., Rajasekar, A. and Wan, M., The SDSC Storage Resource Broker. In *Proc. CASCON'98 Conference*, (1998)
8. Buneman, P., Khanna, S., Tajima, K. and Tan, W.-C., Archiving Scientific Data. In *ACM SIGMOD International Conference on Management of Data*, (2002)
9. Buneman, P., Khanna, S. and Tan, W.-C., Why and Where: A Characterization of Data Provenance. In *International Conference on Database Theory*, (2001)
10. Chen, I.A., Kosky, A.S., Markowitz, V.M. and Szeto, E., Constructing and Maintaining Scientific Database Views. In *9th Conference on Scientific and Statistical Database Management*, (1997)
11. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. and Tuecke, S. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets. *J. Network and Computer Applications* (23). 187-200. 2001.
12. Cui, Y. and Widom, J., Practical Lineage Tracing in Data Warehouses. In *16th International Conference on Data Engineering*, (2000), 367–378
13. Cui, Y., Widom, J. and Wiener, J.L. Tracing the Lineage of View Data in a Warehousing Environment. *ACM Transactions on Database Systems*, *25* (2). 179–227. 2000.
14. Deelman, E., Blackburn, K., Ehrens, P., Kesselman, C., Koranda, S., Lazzarini, A., Mehta, G., Meshkat, L., Pearlman, L., Blackburn, K. and Williams., R., GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists. In *11th Intl Symposium on High Performance Distributed Computing*, (2002)
15. Deelman, E., Foster, I., Kesselman, C. and Livny, M. Representing Virtual Data: A Catalog Architecture for Location and Materialization Transparency. Technical Report GriPhyN-2001-14, 2001, www.griphyn.org.
16. Deelman, E., Kesselman, C. and Mehta, G. Transformation Catalog Design for GriPhyN. Technical Report GriPhyN-2001-17, 2001, www.griphyn.org.
17. Foster, I. and Kesselman, C. A Data Grid Reference Architecture. Technical Report GriPhyN-2001-12, 2001, www.griphyn.org.
18. Foster, I. and Kesselman, C. (eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
19. Frey, J., Tannenbaum, T., Foster, I., Livny, M. and Tuecke, S., Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *10th International Symposium on High Performance Distributed Computing*, (2001), IEEE Press, 55-66
20. Innocente, V., Silvestris, L. and Stickland, D. CMS Software Architecture: Software Framework, Services, and Persistency in High-level Trigger, Reconstruction, and Analysis. *Computer Physics Communications*, *140*. 31-44. 2001.
21. Ioannidis, Y.E. and Livny, M. Conceptual Schemas: Multi-faceted Tools for Desktop Scientific Experiment Management. *International Journal of Cooperative Information Systems*, *1* (3). 451-474. 1992.
22. Ioannidis, Y.E., Livny, M., Gupta, S. and Ponnekanti, N., ZOO : A Desktop Experiment Management Environment. In *22th International Conference on Very Large Data Bases*, (1996), Morgan Kaufmann, 274-285
23. Leymann, F. and Altenhuber, W. Managing Business Processes as an Information Resource. *IBM Systems Journal*, *33* (2). 326–348. 1994.
24. Litzkow, M., Livny, M. and Mutka, M. Condor - A Hunter of Idle Workstations. In *Proc. 8th Intl Conf. on Distributed Computing Systems*, 1988, 104-111.
25. M. Della Negra, S., The CMS Experiment,. A Compact Muon Solenoid. cmsinfo.cern.ch/Welcome.html.
26. Marian, A., Abiteboul, S., Cobena, G. and Mignet, L., Change-Centric Management of Versions in an XML Warehouse. In *27th International Conference of Very Large Data Bases*, (2001)
27. Mohan, C., Alonso, G., Gunthor, R. and Kamath, M. Exotica: A Research Perspective on Workflow Management Systems. *Data Engineering Bulletin*, *18* (1). 19-26. 1995.
28. Stockinger, H., Samar, A., Allcock, W., Foster, I., Holtman, K. and Tierney, B., File and Object Replication in Data Grids. In *10th IEEE Intl. Symp. on High Performance Distributed Computing*, (2001), IEEE Press, 76-86
29. Szalay, A. and Gray, J. The World-Wide Telescope. *Science*, *293*. 2037-2040. 2001.
30. Williams, R., Bunn, J., Moore, R. and Pool, J. Interfaces to Scientific Data Archives. Center for Advanced Computing Research, California Institute of Technology, Technical Report CACR-160, 1998.
31. Woodruff, A. and Stonebraker, M. Supporting Fine-Grained Data Lineage in a Database Visualization Environment. Computer Science Division, University of California Berkeley, Report UCB/CSD-97-932, 1997.
32. Zhao, Y. Virtual Galaxy Clusters: An Application of the GriPhyN Virtual Data Toolkit to Sloan Digital Sky Survey Data. MS thesis, University of Chicago, Technical Report GriPhyN-2002-06, 2002, www.griphyn.org.